JBoss Rules

- Mark Proctor
  - ✓Project Lead

- The SkyNet funding bill is passed.
- The system goes online on August 4th, 1997.
- Human decisions are removed from strategic defense.
- SkyNet begins to learn at a geometric rate.
- It becomes self-aware at 2:14am Eastern time, August 29th
- In a panic, they try to pull the plug.
- And, Skynet fights back

Agenda

- Quick look at a Rule
- A bit more on Rules
- Golfing Configuration Example
- Rule Engine Background
- Deeper look into Rule Engines

What is a Rule

What is a Rule

Quotes on Rule names are optional if the rule name has no spaces.

salience <int>
agenda-group <string>
no-loop <boolean>
auto-focus <boolean>
duration <long>

```
rule "<name>"
    <attribute> <value>
    when
        <LHS>
    then
        <RHS>
end
```

RHS can be any valid java. Future versions will support other languages, i.e Groovy

What is a Rule

Methods that must be called directly

specific passing of instances

```
public void helloMark(Person person) {
    if ( person.getName().equals( "mark" ) {
        System.out.println( "Hello Mark" );
    }

rule "Hello Mark"
    when
        Person( name == "mark" )
    then
        System.out.println( "Hello Mark" );
end
```

Rules can never be called directly

Specific instances cannot be passed.

LHS

RHS

What is a Rule

- Column with no field constraints
  - ✓ Person()
- Column with a literal field constraint
  - ✓ Person(name == "bob")
- Column with field binding
  - ✓ Person($bob : name == "bob")
  - ✓ Variable names can be any valid java variable, $ is optional but helps differentiate between fields and variables
- Column with Fact binding
  - ✓ $bob : Person(name == "bob")
- Column with bound variable constraint
  - ✓ Person(name == $name)
- Column and field bindings are referred to as declarations.

What is a Rule

- More Field Constraints
  - ✓ Return value constraint
    - Person($age age)
      Person(age == ($age + 2)
  - ✓ Predicate value constraint
    - Person($age1 age)
      Person($age2 : age ->
      ($age1.toValue() ==
      $age2.toValue() + 2)
- Conditional Elements
  - ✓ 'and'
  - ✓ 'or'
  - ✓ 'not'
  - ✓ 'exists'
  - ✓ 'eval'

Package

Namespace for all package members

Imports can be used in functions and rules. Uses valid java import syntax

```
package com.sample

import java.util.Map
import com.sample.Cheese

global Cheese cheese

function void exampleFunction(Cheese cheese) {
    System.out.println( cheese );
}

rule "A Cheesy Rule"
    when
        ….
    then
        ….
end
```


A bit more on Rules

Business Rules Approach Methodology

- STEP
  - ✓ **S**eparation of the business rules from components that are not relevant to the business knowledge (mainly in IT systems).
  - ✓ **T**raceability of the business rules, i.e. documenting the sources as well as the usages of every of business rule.
  - ✓ **E**xternalization of the business rules, i.e. making the business rules explicit.
  - ✓ **P**ositioning of the business rules so that it becomes easily changeable and adaptable.

Functional Roles of Rules (RuleSpeak)

- Rejectors
  - ✓ Reject events that cause a violation
- Projectors
  - ✓ "if this then that"
  - ✓ Executes actions
  - ✓ Introduces new Facts
- Producers
  - ✓ Automate computation
  - ✓ Calculates or derives something for the end user

Rejector Rule

```
rule "Underwriting Rule 1_1"
    agenda-group "validation"
    when
        Bike( value > 1000 )
    then
        assert( new Status( "declined" ) );
        list.add( drools.getRule() );
end

rule "Underwriting Rule 1_2"
    agenda-group "validation"
    when
        Bike( value > 400, gears > 10 )
    then
        assert( new Status( "declined" ) );
        list.add( drools.getRule() );
end

rule "Underwriting Rule 2_1"
    agenda-group "validation"
    when
        Bike( value > 200, color == "blue" )
    then
        assert( new Status( "declined" ) );
        list.add( drools.getRule() );
end

rule "Underwriting Rule 2_2"
    agenda-group "validation"
    when
        Bike( color == "green" )
    then
        assert( new Status( "declined" ) );
        list.add( drools.getRule() );
end
```

Projector Rules

Golfing Configuration

Golfing Configuration

There are four Golfers standing at a tea, in a line from left to right.

- The golfer to the Fred's immediate right is wearing blue pants
- Joe is second in line
- Bob is wearing plaid pants
- Tom isn't in position one or four, and he isn't wearing the orange pants

Create all possible combinations

```
String[] names = new String[] {"Fred", "Joe", "Bob", "Tom"};
String[] colors = new String[] {"red", "blue", "plaid",
                                "orange" };

int[] positions = new int[] { 1, 2, 3, 4 };

for ( int n = 0; n < names.length; n++ ) {
    for ( int c = 0; c < colors.length; c++ ) {
        for ( int p = 0; p < positions.length; p++ ) {
            new Golfer( names[n], colors[c], positions[p]) );
        }
    }
}
```


Fred

- The golfer to the Fred's immediate right is wearing blue pants

```
// There is a golfer named Fred,
// Whose positions is $p1
Golfer( $fredsName : name == "Fred",
        $fredsPosition : position,
        $fredsColor : color  )

// The golfer to Fred's immediate right
// is wearing blue pants
Golfer( $unknownsName : name != "Fred",
        $unknownsPosition :  position ==
        ( new Integer( $fredsPosition.intValue() + 1 ) ),
        $unknownsColor : color == "blue",
        color != $fredsColor )
```


Joe

- Joe is second in line

```
// Joe is in position 2
Golfer( $joesName : name == "Joe",
        $joesPosition  : position == 2,
        position != $fredsPosition,
        $joesColor : color != $fredsColor )
```

Bob

- Bob is wearing plaid pants

```
// Bob is wearing plaid pants
Golfer( $bobsName : name == "Bob",
        name != $unknownsName,
        $bobsPosition : position != $fredsPosition,
        position != $unknownsPosition,
        position != $joesPosition,
        $bobsColor : color == "plaid",
        color != $fredsColor,
        color != $joesColor,
        color != $unknownsColor )
```


Tom

- Tom isn't in position one or four, and he isn't wearing the orange pants

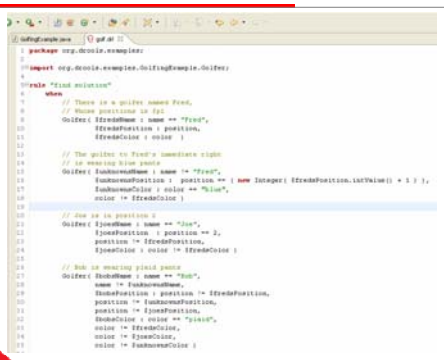
```
// Tom isn't in position 1 or 4
// and isn't wearing orange
Golfer( $tomsName : name == "Tom",
        $tomsPosition : position != 1,
        position != 4,
        position != $fredsPosition,
        position != $joesPosition,
        position != $bobsPosition,
        $tomsColor : color != "orange",
        color != $fredsColor,
        color != $joesColor,
        color != $bobsColor )
```


Results

```
System.out.println( "Fred " + $fredsPosition + " " +
                    $fredsColor );
System.out.println( "Joe " + $joesPosition + " " +
                    $joesColor );
System.out.println( "Bob " + $bobsPosition + " " +
                    $bobsColor );
System.out.println( "Tom " + $tomsPosition + " " +
                    $tomsColor );
```

```
Fred 1 orange
Joe  2 blue
Bob  4 plaid
Tom  3 red
```

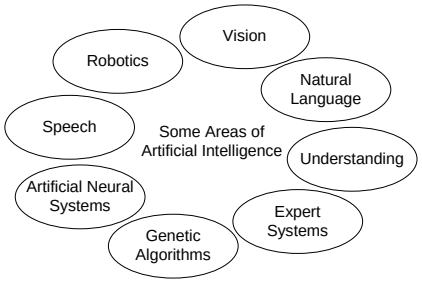

The Eclipse Workbench



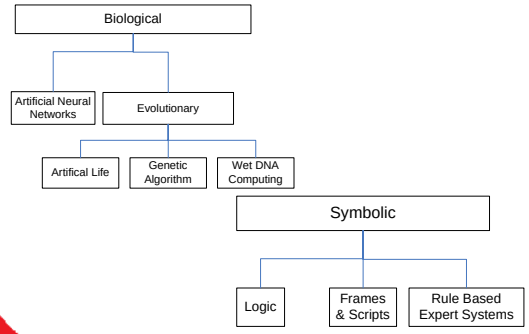
Background

Artificial Intelligence

Making computers think like people

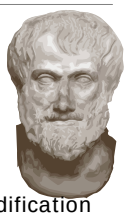


Branches of AI



25

Expert Systems - Knowledge Representation and Reasoning



- The study of Knowledge is Epistemology
- Nature, Structure and Origins of Knowledge

- Expert Systems use Knowledge representation to facilitate the codification of knowledge into a knowledge base which can be used for reasoning
  - ✓ we can process data with this knowledge base to infer conclusions

26

Production Rule System

- JBoss Rules Engine
  - ✓ Rule Based approached to implement an Expert System
  - ✓ correctly classified as a Production Rule System.
- The term "Production Rule" originates from formal grammar
  - ✓ "an abstract structure that describes a formal language precisely"

27

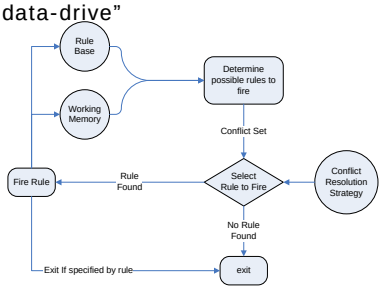
Production Rule System

- Turing Complete
  - ✓ Propositional Logic
  - ✓ First Order Logic
  - ✓ Declarative
- The Brain is the Inference Engine
  - ✓ scale to a large number of rules and facts
  - ✓ matches facts, the data, against Production Rules, also called Productions or just Rules, to infer conclusions which result in actions
  - ✓ A Production Rule is a two-part structure using First Order Logic for knowledge representation.
    - when <conditions> then <actions>
  - ✓ The process of matching the new or existing facts against Production Rules is called Pattern Matching

28

Chaining

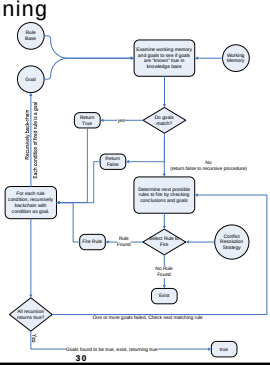
- Forward Chaining
  - ✓ reactionary
  - ✓ "data-drive"



29

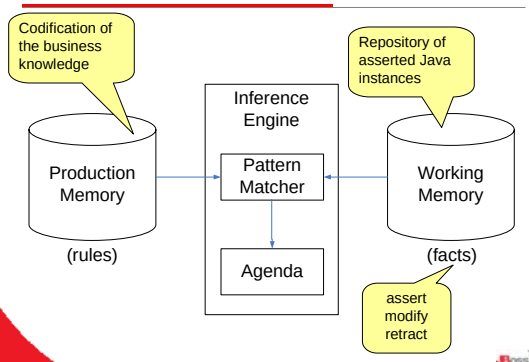
Chaining

- Backward Chaining
  - ✓ "goal-drive"



30

What is a Production Rule System



31

Highlevel Characteristics

- Performance
  - ✓ Rete
- Expressiveness
  - ✓ Declarative rules with proposition and first order logic
- Tooling
  - ✓ Rule editor
  - ✓ Domain Specific Languages
  - ✓ Decision tables
  - ✓ Web authoring
  - ✓ Natural Language Processing

32

Deeper look into Rule Engines

33

Fact Handles

- Fact Handles
  - ✓ Facts are objects a Rule Engine is aware of and reasons over.
  - ✓ Asserted Objects return a handle reference.
  - ✓ The handle is used for modifications and retractions.
  - ✓ Internally the Fact Handle implementation is a long id.
  - ✓ FactHandle handle = workingMemory.assertObject(a);

34

Object Assertion and Pattern Matching

- LHS
  - ✓ One or more Patterns
  - ✓ Patterns are the conditions that must be satisfied for the rule to be legible for firing
- Object assertion
  - ✓ Patterns within the Rule Base are matched. Resulting in partial and full matches for Rules.
  - ✓ Fully matched Rules result in the creation of an Activation
  - ✓ No rules fire at this stage

35

Object Modification

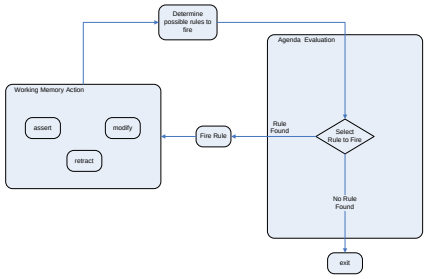
- How to modify a object in the Working Memory
  - ✓ From Java Code
    - workingMemory.modifyObject(factHandle, modifiedFact)
  - ✓ From a Consequence
    - modify(modifiedFact)
- JavaBeans PropertyChangeListeners can provide automatic notification.
- Modifications result in
  - ✓ Activation Cancellations
  - ✓ Activation Creations
  - ✓ Internally this is similar to a retract and assert

36

Two Phase System

- Working Memory Actions
  - ✓ Occurs in Java code and during the execution of a Consequence
  - ✓ Assertion
  - ✓ Deletion
  - ✓ Modification
- Agenda Evaluation
  - ✓ Triggered by Calling workingMemory.fireAllRules()
  - ✓ Executes the first Rule's Consequence and enters Working Memory Action phase. At the end of the Consequence it returns to evaluating the Agenda.
  - ✓ When the Agenda is empty it returns back to the main Java code.

Two Phase System



Tomorrows BOF Session

- Will be delivered using example codes being executed in the Workbench
- Architecture and API
- Workbench views for debugging
- Rule Behaviour
  - ✓ Cross Products
  - ✓ Recursion
- Agenda Groups
- Truth Maintenance
- Temporal Rules

Questions?

- **Dave Bowman**: All right, HAL; I'll go in through the emergency airlock.
- **HAL**: Without your space helmet, Dave, you're going to find that rather difficult.
- **Dave Bowman**: HAL, I won't argue with you anymore! Open the doors!
- **HAL**: Dave, this conversation can serve no purpose anymore. Goodbye.

- **Joshua:** Greetings, Professor Falken.
- **Stephen Falken:** Hello, Joshua.
- **Joshua:** A strange game. The only winning move is not to play. How about a nice game of chess?